面向同步规范的并行代码自动生成

Kai Hu, Teng Zhang, Lihong Shang, Zhibing Yang, Jean-Pierre Talpin

软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(7):1698−1712 [doi: 10.13328/j.cnki.jos.005056]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向同步规范的并行代码自动生成*

胡 凯[1], 张 腾[1], 尚利宏[1], 杨志斌[2,3], Jean-Pierre TALPIN[4]

[1](软件开发环境国家重点实验室(北京航空航天大学 计算机学院),北京 100191)

[2](南京航空航天大学 计算机学院,江苏 南京 210016)

[3](Toulouse Institute of Computer Science Research, Toulouse, France)

[4](Institut National de Recherche en Informatique et en Automatique (INRIA) Rennes, Rennes, France)

通信作者: 尚利宏, E-mail: shanglh@buaa.edu.cn, 张腾, E-mail: rahxephon89@163.com

摘 要: 随着对安全攸关实时系统功能与非功能要求的日益增加,使用多核技术将成为发展趋势.如何在多核平台条件下保证系统运行的可信任性及可靠性是学术上和应用上的关键问题.目前基于形式化方法的系统设计、验证以及自动代码生成已在单核平台上形成很多研究成果,但在多核平台上的研究仍面临许多科学问题.同步语言 SIGNAL 是一种被广泛应用于安全攸关实时系统功能设计的形式化方法,适用于对系统确定性并发行为的描述.SIGNAL 编译器也支持将同步规范(synchronous specification)生成仿真代码,以对其进行验证与分析.然而,现有研究较少关注从 SIGNAL 同步规范到支持跨平台并行代码的生成方法.研究了面向 SIGNAL 同步规范的并行自动代码生成方法.提出了方程依赖图 EDG 的概念,将 SIGNAL 规范转换为 EDG 以分析其全局数据依赖关系;研究了对 EDG 进行任务划分获取规范中可以并行执行部分的算法;最后,以跨平台并行编程 API-OpenMP 作为对象,结合程序中信号的时钟关系,将并行任务映射到 OpenMP 并行代码,并进行了实例验证.

关键词: 同步规范;SIGNAL;并行程序;代码生成;OpenMP

中图法分类号: TP311

中文引用格式: 胡凯,张腾,尚利宏,杨志斌,Jean-Pierre Talpin.面向同步规范的并行代码自动生成.软件学报, 2017,28(7):1698−1712. http://www.jos.org.cn/1000-9825/5056.htm

英文引用格式: Hu K, Zhang T, Shang LH, Yang ZB, Jean-Pierre T. Parallel code generation from synchronous specification. Ruan Jian Xue Bao/Journal of Software, 2017,28(7):1698−1712 (in Chinese). http://www.jos.org.cn/1000-9825/5056.htm

Parallel Code Generation from Synchronous Specification

HU Kai[1], ZHANG Teng[1], SHANG Li-Hong[1], YANG Zhi-Bin[2,3], Jean-Pierre TALPIN[4]

[1](State Key Laboratory of Software Development Environment (School of Computer Science and Engineering, Beihang University), Beijing 100191, China)

[2](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

[3](Toulouse Institute of Computer Science Research, Toulouse, France)

[4](Institut National de Recherche en Informatique et en Automatique (INRIA) Rennes, Rennes, France)

Abstract: As functional and non-functional requirements on safety critical real time systems stack up, the development of multi-core technology in these systems has become a trend. How to guarantee the credibility and reliability on the multi-core platform, however, is the key problem in both academic and industry. While many theoretical and applied achievements have been accomplished on the

* 基金项目: 国家自然科学基金(91538202, 61672074); 软件开发国家重点实验室基金(SKLSDE-2016ZX-16)

Foundation item: National Natural Science Foundation of China (91538202, 61672074); State Key Laboratory of Software Development Environment of China (SKLSDE-2016ZX-16)

收稿时间: 2014-06-23; 修改时间: 2014-12-22, 2015-01-21; 采用时间: 2015-12-28; jos 在线出版时间: 2016-05-03

CNKI 网络优先出版: 2016-05-04 08:44:09, http://www.cnki.net/kcms/detail/11.2560.TP.20160504.0844.004.html

single-core platform, there are still a lot of scientific problems need to be solved on the multi-core platform. Suitable for describing concurrency behaviors, synchronous language SIGNAL is a formalism widely used in the functional design of safety critical real time systems. The SIGNAL compiler supports generating the simulation code from the synchronous specification to verify and analyze the properties of the system model. However, existing studies pay less attention to the generation of multi-platform parallel simulation code from SIGNAL specification. This paper proposes a methodology for automatically generating parallel code from SIGNAL specifications. First, equation dependency graph (EDG) is defined and the specification is translated to analyze the global data dependency relations. Then EDG is partitioned to explore the parallelism of the specification. Finally, altogether with clock relations, parallel tasks are mapped into OpenMP structures. A case study is provided to illustrate the proposed methodology.

Key words: synchronous specification; SIGNAL; parallel program; code generation; OpenMP

安全攸关实时系统(safety critical real time system)[1]应用于航空电子、航天器、汽车控制等领域.由于这些系统如果无法满足设计所要求的正确性,可能导致重大财产损失、人身伤害或者大规模环境损坏等灾难性的后果,因此常用各种形式化方法对其进行严格的设计、开发和验证,形式化方法是一类基于严格数学模型的软/硬件设计方法.系统通过具有非二义性的形式语言进行建模描述,并通过相应的验证工具(自动化模型检测工具或半自动的定理证明器)对模型的各种属性进行验证,在设计早期及时发现系统的设计问题并进行改正.经过验证的模型进一步通过自动代码生成技术转换为可执行代码,形成从设计到最终代码生成的完整过程,从而保证系统的可信性.目前,安全攸关实时系统对于功能及非功能的要求越来越高,传统的单处理器体系结构技术面临着瓶颈,使用多核技术将成为必然的趋势.学术界与工业界已提出了多种并行编程模型及技术,包括 MPI[2]、OpenMP[3]以及 TBB[4]等以支持对多核并行应用.但是,由于多核并行模型的复杂性,在形式化方法系统设计周期中如何实现并行自动代码生成成为一个具有挑战性的难题.

同步规范采用同步假设(synchronous hypothesis)作为其理论基础,将连续的物理时间抽象为离散的逻辑瞬间(instant)的序列.在反应式系统中,系统不断与外界环境进行的交互,被抽象为“输入-计算-输出”这 3 个动作.同步假设理论将这 3 个动作抽象为在一个逻辑瞬间内完成,系统的行为可以被描述为一系列不重叠的动作的序列.通过使用同步规范,设计人员可以对系统的功能行为进行描述而不需要考虑具体的平台.

在此基础上,产生了广泛用于安全攸关实时系统计的形式化描述语言——同步语言(synchronous language)[5,6].目前,主流的同步语言包括 Esterel[7],Lustre[8],SIGNAL[9]以及 QUARTZ[10]等.而其中 SIGNAL 由于具有多时钟(multi-clock)的特点,并且采用数据流方程的组合作为建模形式,因此适合描述分布式或并发系统,并已成功应用于空客 A350 的舱门控制系统中[11].同时,现有的 SIGNAL 编译器 Polychrony[2]提供了将同步程序自动生成仿真代码的能力,可以在程序执行层面对模型的功能设计作进一步验证.然而现有编译器实现及相关研究较少关注对面向 SIGNAL 的并行自动生成方法的研究.因此,本文选取跨平台并行编程技术 OpenMP 作为生成对象,研究并提出面向 SIGNAL 同步语言的并行仿真代码的自动生成方法.

本文第 1 节对同步语言 SIGNAL 的语法及语义进行简要介绍.第 2 节介绍基于方程依赖图 EDG(equation dependency graph)的 OpenMP 并行代码生成方法.第 3 节通过一个 SIGNAL 程序实例对本文提出的方法作进一步说明.第 4 节对 SIGNAL 代码生成的研究现状进行介绍.最后一节给出本文总结并展望今后进一步的工作.

1 同步理论及 SIGNAL 简介

1.1 同步理论

根据对系统设计中时间表示的抽象程度,实时系统编程模型分为 3 类:异步模型、预估时间模型以及同步模型[13,14].而其中同步模型与异步模型和预估时间模型最大的不同是时间被高度抽象为离散的序列,一次反应(reaction,即输入-计算-输出)的执行时间被定义为可在一个逻辑瞬间内完成,不需要考虑执行的物理时间,因此,同步模型是一种平台无关模型.本节将对同步理论的相关知识及术语进行简要介绍.

1.1.1 同步假设

同步假设(synchronous hypothesis)[15]是同步语言所依赖的最基本的假设:对于每一个输入,系统都会在下一

次输入之前完成计算并进行输出.这个假设保证了每一个反应之间不会出现重叠.因此,同步模型不考虑时间的量化信息,而是关注事件的到达顺序与事件间的同步.这种特点使得用户可以在不关心系统的具体时间属性的前提下,处理系统的时序问题.下面对下文将要使用的同步模型相关术语进行说明[16].

(1) 逻辑瞬间与反应

同步模型中系统时间被抽象为不重叠的逻辑瞬间(instant)的序列 $tn(n\in N+)$.在每个瞬间,输入信号被读入,进行计算,并得到输出,系统进入一个新的全局状态.而每个瞬间的执行被称为反应(reaction).

(2) 信号

信号(signal)是一个由值到离散序列的映射 $V\rightarrow tn(n\in N+)$.其中 V 除了包含信号的数据类型外,也包括值"⊥".在某一逻辑瞬间,信号值可以存在(present)或缺位(absent),当缺位时,信号值被定义为⊥.

(3) 抽象时钟

信号的抽象时钟(abstract clock,简称为时钟)是离散序列中值存在的逻辑瞬间的集合.如果某一逻辑瞬间属于信号的时钟,那么在此逻辑瞬间可以对此信号进行读取或计算.

1.1.2 单时钟与多时钟模型

安全攸关实时系统往往是由多个组件或子系统组成,甚至有可能被部署在分布式环境中.在设计此类系统时,有两种观点,单时钟(monoclock)与多时钟(multiclock).在单时钟模型中,整个系统中的所有行为触发都由一个全局时钟(master clock)所控制.其优点是对系统功能行为的描述更加简单,并且容易生成仿真代码.然而,在单时钟模型中,各个组件的时钟将与系统全局时钟有严格的隶属关系,这将导致子系统的时钟间产生紧耦合,一旦一个组件的时钟频率出现变化,全局时钟以及其他组件的时钟也需要进行相应的调整.Esterel 及 Lustre 采用单时钟模型.

与单时钟模型不同,多时钟模型没有全局时钟,各个组件都按照自己的时钟工作.组件之间是松耦合的,时间同步操作只需在进行交互的组件之间进行.因此,多时钟模型适合对分布式系统或具有高度并行性的系统进行建模.但是,多时钟会带来模型中信号间同步的复杂性,一方面,模型需要进行时钟一致性的验证,另一方面,也增加了仿真代码生成的难度.SIGNAL 采用多时钟模型.

1.2 SIGNAL简介

SIGNAL 是由法国 CNET 及 INRIA 发明的一种声明式数据流同步语言.与 Esterel 和 Lustre 等其他同步语言不同,SIGNAL 在最初设计时就采用了多时钟的计算模型,被称为 Polychrony[17].信号是 SIGNAL 的基本操作对象,被定义为无限长度的带有类型的值序列$(x_t),t\in N$.给定一个逻辑瞬间,信号可以存在并带有一个值,或者缺位,记作⊥.时钟则被用于表示信号在每个逻辑瞬间是否存在,记作^x.在 SIGNAL 中,时钟为 event 类型的信号:存在时值为真,否则缺位.系统行为则通过描述信号或时钟间的关系的数据流方程(dataflow equation)加以指定.

SIGNAL 提供 4 种基本结构(primitive construct)以描述数据流方程,包括瞬时关系(instantaneous relation)、延迟(delay)、欠采样(undersampling)以及确定性合并(deterministic merging,以下简称合并).表 1 展示了 4 种基本结构的语法及基本语义.需要说明的是:(1) 瞬时关系操作中的 f 可以为代数运算或布尔运算;(2) 由于延迟操作的特殊含义,在下文也被称为记忆方程(memory equation),而方程的左值信号也被称为记忆信号或状态信号.

Table 1 Syntax and semantics of primitive constructs on dataflow equations

表 1 数据流方程基本结构语法及语义

名称	语法	语义
Instantaneous relations	$y:=f(x_1,x_2,\ldots,x_n)$	当 $x_1,\ldots,x_n$ 同时存在时,y 的值为 $f(x_1,x_2,\ldots,x_n)$;否则 y 缺位
Delay	$y:=x$ $ init c	y 的取值为 x 在前一个逻辑瞬间的值,初值为 c,当 x 缺位时,y 缺位
Undersampling	$y:=x$ when z	当 z 存在取值为 true 并且 x 存在时,y 的值为 x 的值;否则 y 缺位
Deterministic merging	$y:=x$ default z	当 x 存在时,y 的取值为 x;当 x 不存在而 z 存在时,y 的取值为 z;否则 y 缺位

SIGNAL 的基本结构除了表示了信号间的数据依赖关系外,也隐含了信号间的时钟关系,见表 2.对于瞬时关系,方程隐含右值和左值所有信号需要有相同的时钟,即信号是同步的.延迟操作也有相同的时钟关系.这两

种操作也被称为单时钟操作(monoclock operator).对于欠采样,信号 y 的时钟为真当且仅当 x 的时钟为真、b 的时钟为真并且 b 的值为真(记作[b]).对于合并操作,信号 y 的时钟为信号 x 与 z 时钟的并集.由于后两种操作中信号的时钟可以不同,因此被称为多时钟操作(multiclock operator).

Table 2 Primitive constructs and corresponding clock relations

表 2 基本结构与相应的时钟关系

语法	时钟关系	解释
$y=f(x_1,x_2,\ldots,x_n)$	$\hat{}y=\hat{}x_1=\ldots=\hat{}x_n$	方程中的所有信号时钟同步
$y:=x$ $ init c	$\hat{}y=\hat{}x$	右值信号 x 与左值信号 y 同步
$y:=x$ when b	$\hat{}y=\hat{}x\cap[b]$	当 b 存在并且取值为 true 并且 x 存在时,y 存在
$y:=x$ default z	$\hat{}y=\hat{}x\cup\hat{}z$	信号 y 的时钟为 x 与 z 时钟的并

SIGNAL 中数据流方程表示了信号或时钟间的关系,而 SIGNAL 程序的基本单位被称为进程(process).它是由一组数据流方程组成的.SIGNAL 在 process 也定义了两个基本结构:组合(composition)与局部声明(local declaration).表 3 给出了组合和局部声明的基本语法与语义.

Table 3 Syntax and semantics of primitive constructs on process

表 3 进程中基本结构的语法及语义

名称	语法	语义
Composition	$P\|Q$	P 和 Q 为进程;$P\|Q$ 的行为是 P 和 Q 行为的组合
Local declaration	P where t_1 s_1; t_2 s_2;…t_n s_n; end;	P 为进程,s_1 到 s_n 为定义在 P 中的信号,即在进程 P 外不可见

在此可以得到 SIGNAL 基本语法的 BNF 表达,其中,数据流方程 $x:=yfz$ 表示数据流方程;$P|Q$ 为进程间的组合;P/x 为进程中的局部声明:

$$P,Q ::= x := yfz \mid P \mid Q \mid P/x.$$

SIGNAL 还提供了时钟操作符"^"以及记忆操作符"cell"等扩展操作符(extended construct),以简化 SIGNAL 程序的表达.但所有扩展操作符都可以通过基本操作符进行表示,因此本文中的 SIGNAL 程序都是使用基本语法写成的.更多关于 SIGNAL 语法的介绍可参见文献[18],关于 SIGNAL 形式语义的研究可参见文献[19].

2 基于 EDG 的并行代码生成方法

SIGNAL 通过数据流方程组合的形式描述系统的功能行为,没有数据和时钟依赖关系的方程可以并行执行.本文提出一种基于方程依赖图(equation dependency graph,简称 EDG)以发现程序中隐含的并行执行部分的方法,并以此为基础生成并行仿真代码.与文献[20]提出的方法相比,本文将 SIGNAL 规范中的时钟信息考虑在内.总体方法流程如图 1 所示.SIGNAL 规范包含了信号间的数据依赖并隐含了信号时钟间的关系.通过数据依赖分析可以得到方程依赖图 EDG;通过分析时钟关系可以得到规约范式方程集合 RNFS.之后,通过对 EDG 进行并行任务划分并加入时钟关系信息,可以得到任务序列.任务序列将最终被映射到 OpenMP 结构以得到并行仿真代码.

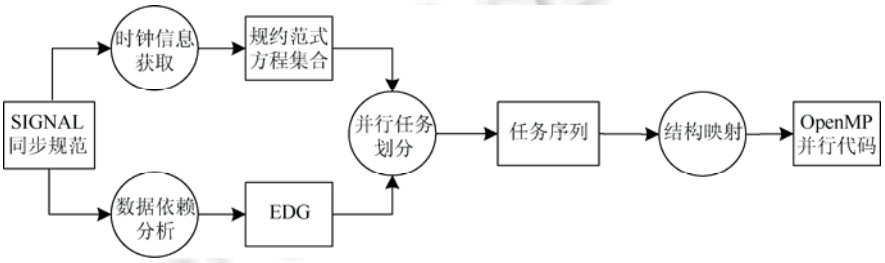

Fig.1 Process of parallel code generation from SIGNAL specification

图 1 面向 SIGNAL 规范的并行代码生成流程

2.1 时钟信息的获取

SIGNAL 中,信号在其时钟取值为真的逻辑瞬间被计算(或被读入,如果信号为输入信号),其一般的代码生成模式为"**if** ^x **then** compute(x)",表示当且仅当信号 x 的时钟为真,则可以对 x 的值进行计算.因此,为了生成正确的代码,需要获取信号间的时钟关系,即时钟如何被计算的信息.本节将对 SIGNAL 程序时钟信息的分析获取过程进行简要说明.

(1) 时钟定义的获取

根据 4 种基本结构隐含的时钟信息,SIGNAL 程序的数据流方程描述了信号间的时钟关系.由于在 SIGNAL 中,时钟由 event 类型信号表示,因此时钟间的关系方程可以看作布尔方程,被称作时钟方程.通过对时钟方程集合的解析,可以得到每个信号时钟的唯一定义,被称为范式方程集合.在解析过程中,如果发现一个信号有多个不等价的定义,则无法通过此 SIGNAL 程序生成确定性的仿真代码.

(2) 时钟等价类的划分及规约范式方程的获取

SIGNAL 程序中,两个信号时钟间同步关系满足自反性、对称性与传递性,因此是定义在时钟集合 C 上的等价关系.从而可以根据同步关系对时钟集合进行划分 $C=\{C_i|i\in Z+\}$.对于任意 C_i,其包含元素$^c_1,\ldots,^c_k$ 是信号对应的时钟变量,并且相互之间同步.将同步的时钟进行划分并进行合并可以有效地减少需要计算的时钟数量,提高仿真代码的执行效率.在文献[21]中,这种划分被称为时钟等价类(clock equivalence class).根据上下文语境,下文中时钟等价类也将被称为时钟.

通过遍历范式方程集合,可以将具有等价时钟定义的信号划分到一个时钟等价类中.之后将时钟方程中的时钟变量替换为其所在的时钟等价类即可获得表达时钟等价类间关系的方程,被称为规约范式方程(reduced normal form,简称 RNF),其集合被称为 RNFS.需要说明的是,为了生成具有确定性的仿真代码,SIGNAL 程序需满足 endochrony 属性[22],即程序中仅能存在一个未被定义的根时钟,其他时钟定义都将通过根时钟派生出来.本文假定生成的 RNFS 已经包含了除一个根时钟以外所有的时钟的定义,而其包含的时钟信息也将被用于仿真代码的生成.在文献[23]中,我们提出了包括 RNF 定义在内的时钟信息获取的详细方法.

2.2 方程依赖图 EDG 的定义及构造方法

为了支持对 SIGNAL 的代码生成,文献[24]最早提出了同步流数据依赖图(synchronous flow dependency graph,简称 SFDG)的概念.SFDG 包含有向无环图(DAG),DAG 中的节点为信号,边为信号间的依赖关系.节点和边的出现则由根据定义在其上的激活时钟所决定.然而,SFDG 中没有包含关于信号如何根据数据依赖进行计算的信息.为了更好地描述方程之间的依赖关系并方便最终的映射,本文提出方程依赖图 EDG 这一概念.EDG 将数据流方程作为图中的节点,边表示为方程间的数据依赖关系.这里,首先给出 EDG 中节点(记作 NDG)的定义.

定义 1. EDG 节点 NDG 是一个三元组$\langle B,L,R\rangle$,其中,B 是一个二元组$\langle G,A\rangle$,包含守卫条件 G 和动作 A,表示的含义为当 G 为真时,执行动作 A(A 在后文也将被称为信号定义方程);L 为动作 A 的左值信号;R 为动作 A 的右值信号.

基于以上定义,表 4 给出对于 4 种基本结构及输入信号到对应 NDG 的映射规则.

Table 4 Mapping from primitive constructs to NDG

表 4 基本结构到 NDG 的映射

语法	L	R	G
$y=f(x_1,x_2,\ldots,x_n)$	y	$x_1,\ldots,x_n$	**if** C_y **then** $y:=f(x_1,\ldots,x_n)$
$y:=x$ \$ init c	NULL	NULL	NULL
$y:=x$ when b	y	x,z	**if** C_y **then** $y:=x$
$y:=x$ default z	y	x	**if** C_x **then** $y:=x$
		z	**if** $C\_z\backslash C\_x$ **then** $y:=z$
输入信号 s	S	NULL	**if** C_s **then** read(s)

对于瞬时关系,y 对应于节点的 L;右值集合 $x_1,\ldots,x_n$ 对应于节点的 R;B 中的 G 为信号 y 的时钟 C_y,动作 A

为关系操作 $y:=f(x_1,\ldots x_n)$.由于 y 以及 x_1 到 x_n 的信号是同步的,当 y 的时钟为真时,x_1 到 x_n 一定存在.

对于延迟操作,由于信号 y 和 x 之间没有数据依赖关系,因此不生成对应的 NDG 节点.但是记忆信号 y 可以作为其他节点的右值.

对于欠采样操作,y 对应于节点的 L;右值 x 与 z 对应于节点的 R;B 中的 G 为信号 y 的时钟 C_y,动作为赋值操作 $y:=x$.可以看出,若要动作 $y:=x$ 得到执行,需要保证 x 和 z 的值已经得到,并且 C_y 的值为真.

对于合并操作,将生成两个节点,分别对应于 y 对 x 及 z 值的选择.当 x 的时钟 C_x 为真时,y 将被赋值为 x;当 x 的时钟为假并且 z 的时钟为真时,y 将被赋值为 z.由于两个节点执行动作的守卫条件是互斥的,因此不需要考虑共享变量更新的问题.

对于输入信号的 NDG,其中,由于输入信号从环境中取值,因此 R 为空.当 L 值信号时钟为真时,做读取操作.

EDG 实质上是一种包含 SIGNAL 程序全局数据依赖关系的有向无环图,这里给出方程依赖图 EDG 的定义.

定义 2. EDG 为一个二元组$\langle SNDG,\rightarrow\rangle$,其中,$SNDG$ 为节点 NDG 的集合;$\rightarrow$为节点之间的依赖关系:若节点 v_1,v_2 满足二元关系 $v_1\rightarrow v_2$ 当且仅当 $v_1.L\in v_2.R$.

通过分析数据流方程集合可以得到 $SNDG$,其中,每个节点中守卫条件时钟由对应的时钟等价类表示.EDG 则通过分析 $SNDG$ 之间左右值的数据关系得到.需要说明的是,对于方程右值集合中出现记忆信号(即延迟操作左值信号)的情况,由于延迟操作特殊的语义,可以保证当节点中的其他数据依赖以及守卫条件 G 得到满足时,从记忆信号得到的值也是正确的.

2.3 基于 EDG 的并行任务划分方法

通过分析 EDG 获得方程间的数据依赖关系,并将其划分为可以并行执行以及必须串行执行的部分.首先,将通过拓扑排序的方法对 EDG 进行并行任务划分;之后,将时钟信息加入到并行任务中.

2.3.1 基于拓扑排序的 EDG 划分

在 EDG 的定义中,二元关系"$\rightarrow$"定义了节点间的依赖关系,即对于两个节点 v_1、v_2,如果 v_1 对应信号定义方程 eq_1 的左值信号出现在 v_2 对应信号定义方程 eq_2 的右侧,则说明 eq_2 依赖 eq_1.由于数据依赖关系满足严格偏序关系,可以定义其传递闭包,记作"$\searrow$".若程序的两个数据流方程 eq_1 和 eq_2 之间不存在此关系,这两个方程可以并行执行.为了从 SIGNAL 程序生成并行仿真代码,需要 EDG 作进一步划分.在此,我们给出并行任务(parallel task)的定义.

定义 3. 并行任务为一个二元组$\langle T,\nearrow\rangle$,其中,

- T 是对 EDG 中节点集合的划分,即(1) 对于任意 $t\in T,t\subseteq EDG.SNDG$;(2) 对于任意两个节点 t_1、$t_2\in T,t_1\cap t_2=\varnothing$;(3) T 中所有元素的并集为 EDG 的节点集合,我们称元素 t 为任务节点(task node).
- $\nearrow$是定义在 T 上的二元关系,称为优先关系(precedence relation).

目标平台的不同会导致并行任务划分方法对最终程序的运行性能产生很大的影响.本文采取拓扑排序的方法 EDG 进行划分.其基础是定义划分 T 和优先关系的性质.

- 对于任意 $t\in T,t$ 是定义在$\searrow$上的反链(anti-chain),即任意两个节点 n_1、$n_2\in t,n_1\searrow n_2$ 以及 $n_2\searrow n_1$ 恒为假.
- 对于任意 $t_1,t_2\in T,t_1\nearrow t_2$ 当且仅当 t_2 中至少存在一个节点 n 使得 t_1 中存在一个节点 m,且有 $m\searrow n$.

图 2 给出了 EDG 划分算法.算法输入为 EDG,输出为表示并行任务的有序列表 TaskList.首先将 EDG 中的所有节点加入到 NodeSet 集合中(第 3 行),之后进入迭代过程(第 4 行).第 5 行~第 10 行描述了拓扑排序的过程:首先找到 EDG 中没有入度的节点(第 5 行),这里的入度是针对数据依赖关系"$\rightarrow$"而言的.获得的无入度的节点集合 setNoDegree 将组成 Task 中的一个元素,记作 tempTask(第 6 行).将 tempTask 加入到 TaskList 后(第 7 行),需要将 setNoDegree 中所有节点以及以其为出度的二元关系从 NodeSet 集合中移除(第 9 行及第 8 行).迭代的结束条件是 NodeSet 为空,即图中所有节点都已被划分到 TaskList 中.

对于任意 EDG 图,经拓扑排序得到的序列 TaskList=$\{t_{n1},t_{n2},\ldots,t_{nk}\}$,满足以下性质.

- 任意 $t\in T$ 是定义在$\searrow$上的反链.
- 对于任意整数 $x,y\in\{n_1,\ldots,n_k\}$,t_x 和 t_y 满足关系 $t_x\nearrow t_y$ 或 $t_y\nearrow t_x$.

● 对于任意两个整数 $n_x,n_y,x,y\in\{1,\ldots,k\}$,若 $x<y$,对于任意节点 $n\in t_{ny}$,不存在节点 $m\in t_{nx}$,使得 $n\searrow m$ 成立.

得到的任务序列 TaskList 是并行任务中的一种特殊情况,优先关系↗为任务节点间的全序关系.这里,进一步定义 TaskList 序列间的相邻关系:对于任务序列 TaskList=$\{t_{n1},t_{n2},\ldots,t_{nk}\}$以及 $x,y\in\{n_1,\ldots,n_k\}$,若 $y=x+1$,则称任务节点 t_x 左相邻于任务节点 t_y,记作 $t_x\Rightarrow t_y$.

以上算法适用于 EDG 中没有出现循环依赖的情况.若存在信号 s 满足 $s\searrow s$,则无法对 EDG 进行划分.需要说明的是,多时钟操作符中不同时钟下的数据依赖可以不同.例如,对于合并操作 $c:=d$ default e,在 d 的时钟为真时,c 依赖于 d;而在 d 的时钟为假而 e 的时钟为真时,c 依赖于 e.因此,对于数据依赖链 $n_1\xrightarrow{c_1}n_2\xrightarrow{c_2}\ldots\xrightarrow{c_{k-1}}n_1$,若有 $c_1\wedge c_2\wedge\ldots\wedge c_{k-1}$ 的值恒为 0,则循环依赖条件永远不会成立,被称为伪循环依赖,本文中则假设程序没有循环依赖或伪循环依赖.

```
1:Input: EDG
2:Output:TaskList%有序表,先后顺序表示了线性关系
3:NodeSet←EDG.Nodes%初始值 NodeSet 包含了 EDG 中的所有节点
4:while NodeSet≠∅
5:      setNoDegree←findNodeWithNoIndegree(NodeSet)%找到 NodeSet 中没有入度的节点
6:       tempTask←createT(setNoDegree)%建立 Task 中的元素 t
7:       addTask(TaskList,tempTask)%将新建元素加入到 TaskList 的末尾
8:       removeRelation(NodeSet,setNoDegree)%删除与 setNoDegreee 集合中节点相关的二元关系
9:       removeNode(NodeSet,setNoDegree)%从 NodeSet 删除 setNoDegree 中的所有节点
10:endwhile
11:return TaskList
```

Fig.2 Partition of EDG based on topological sorting

图 2 基于拓扑排序的 EDG 划分算法

2.3.2 时钟信息与并行任务的合并

通过对 EDG 进行任务划分,可以得到任务序列 TaskList,其元素将根据序列顺序执行,而元素内部的节点由于不存在数据依赖关系可并行执行.然而,此结构中缺少对时钟如何进行计算的信息,因此需要将规约范式方程集合 RNFS 中的时钟定义方程放入到任务序列中正确的位置.在此,首先给出需满足的性质.

● 所有信号定义必须在其激活时钟被计算后执行.

● 所有时钟定义必须在方程右值时钟或取值变量被计算后执行.

为了满足上述性质,本文通过遍历 TaskList,在其二元相邻关系"⇒"中加入时钟定义方程集合,记作 $t_1\overset{\text{ClockDef}}{\Rightarrow}t_2$. 其中,ClockDef 为任务 t_1 和 t_2 之间的时钟定义方程集合.表示的含义是任务 t_2 执行前必须首先执行 ClockDef 中的方程;而 ClockDef 中的方程可能将依赖 t_1 中的计算结果.

由 endochrony 性质可知,根时钟的值恒为真,因此,从 TaskList 的第 2 个元素开始进行有序遍历.首先,获取当前 task 中的信号定义;之后从 RNFS 找出之前没有被加入到二元关系中并依赖这些信号定义方程的 RNF 集合 ClockDef;最后将 ClockDef 加入到二元关系 task.pre⇒task 之间,形成 task.pre $\overset{\text{ClockDef}}{\Rightarrow}$ task.

对于一个任务序列 $t_1\Rightarrow t_2\Rightarrow t_3\ldots\Rightarrow t_n$,我们可以得到一个序列 $t_1\overset{\text{ClockDef1}}{\Rightarrow}t_2\overset{\text{ClockDef2}}{\Rightarrow}t_3\overset{\text{ClockDef3}}{\Rightarrow}\ldots\overset{\text{ClockDefn-1}}{\Rightarrow}t_n$ 满足以下性质.

● 对于任意二元关系 $t_k\overset{\text{ClockDefk}}{\Rightarrow}t_{k+1}$,ClockDefk 中的方程计算不依赖于节点 $t_{k+1},\ldots,t_n$.

● 任意 t_k 中信号定义方程需要的时钟 $C_1,\ldots,C_m$ 的值一定已经在时钟定义方程 ClockDef1,…,ClockDef$k$–1 中被计算出来.

以上性质将保证序列保持 SIGNAL 程序的数据依赖关系以及时钟关系.

2.4 并行任务到 OpenMP 并行结构的映射

OpenMP 是一种多平台的共享内存的并行编程技术,所支持的编程语言包括 C、C++以及 FORTRAN 等.它提供了包括编译制导、应用编程接口以及环境变量等多种机制,支持用户在高层次对并行算法进行描述.前文已将 SIGNAL 程序转换为并行任务序列,并且执行动作均为简单数学、逻辑计算或赋值.因此,本文选择编译制导语法 parallel sections 作为映射的对象,其语法如下所示.

```
#pragma omp sections [clause[[,] clause] ...]
{
[#pragma omp section]
structured-block
[#pragma omp section ]
structured-block
...
}
```

制导语句#pragma omp sections 所包含的块内为一个并行域.而由制导语句#pragma omp section 包含的代码块 structured-block 之间是并行执行的,代码块内部则是串行执行.根据并行任务序列的语义,可以得到从任务序列元素到 sections 语法的映射规则,见表 5.

Table 5 Mapping from TaskList to sections

表 5 任务序列元素到 sections 的映射

任务序列元素	sections 语法
任务节点	#pragma omp sections{ }

信号定义方程 eq	#pragma omp section{
	if(eq.Class==true){
	eq.A
	}}

任务序列 $t_1 \Rightarrow t_2 \Rightarrow t_3 \ldots \Rightarrow t_k$	#pragma omp sections{
	t1 %t1 对应的 sections 代码块
	}
	…
	#pragma omp sections{
	tk %tk 对应的 sections 代码块
	}

任务序列中的每个任务将被映射为一个由制导指令#pragma omp sections 所包围的代码块.而任务中每个节点的信号方程将被映射为一个由制导指令#pragma omp section 所包围的代码块.根据 sections 的语义,所有处于一个任务内的信号定义方程将可以并行执行.表中第 3 行表示任务序列的顺序执行关系将映射为 sections 代码块间的顺序执行关系.此外,在 OpenMP 编译中,并行任务需要编译器开启多个线程,而多线程的操作需要消耗时间及资源.因此若一个任务中只含有一个节点,则不再生成 sections 以及 section 代码块,而直接生成对应的动作,以提高程序的效率.

除了任务以及任务节点以外,在映射中还需要处理对时钟定义方程的映射.时钟定义方程本质上也是赋值运算,因此可以采用相似的映射方法.为了增加程序的并行度,对于任务序列中的二元关系 $t_1 \overset{\text{ClockDef1}}{\Rightarrow} t_2$,首先对时钟定义方程集合 ClockDef1 进行处理.

- 若 ClockDef1 中只含有一个方程,则直接生成对应的赋值动作.
- 若 ClockDef1 有多个方程,并且方程间不存在数据依赖关系,则将此方程集合也看作一个任务,采用相同的方法生成 sections 以及 section 代码块.
- 若多个方程间存在数据依赖关系,则根据数据依赖关系生成串行执行序列.

通过以上映射方法,可以得到满足 SIGNAL 程序数据及时钟依赖关系并且具有一定并行度的 OpenMP+C 仿真代码.在对目标平台的特点(如处理器核数)以及 OpenMP 编译器实现作进一步了解后,可以进一步对映射进行优化,例如根据 section 代码块的个数设定生成线程数以及 section 间代码的合并等等.

本文采用 SIGNAL 编译器 Polychrony 所使用的迭代模式对 SIGNAL 程序的行为进行仿真:无限循环模拟程序与环境的交互过程,每次循环代表一个逻辑瞬间中的执行过程.根据前文提出的方法得到的 OpenMP 并行结构将作为迭代的核心部分,而对记忆信号值的更新则被放在每次迭代结束之后进行.

3 实例分析

本节将以一个典型的 SIGNAL 程序为例介绍生成 OpenMP 并行仿真代码的过程.首先,将通过分析数据流

方程获取时钟信息;之后通过数据依赖分析生成 EDG 并进行任务划分;最终将并行任务序列映射到 OpenMP 并行结构中.

3.1 示例 SIGNAL 程序

本文选择同步语言中经典示例程序 ABRO[25]作为转换实例 ABRO 自动机表示,如图 3 所示.ABRO 有 3 个输入信号,A、B、R 以及一个输出信号 O.在初始状态,ABRO 等待输入信号,当 A、B 信号按照任意次序读入后,ABRO 输出信号 O.信号 R 用于重置 ABRO 的状态.根据 ABRO 的功能特点,文献[16]给出了相应的 SIGNAL 程序的 SIGNAL 程序,本文给出的修改版本如图 4 所示.

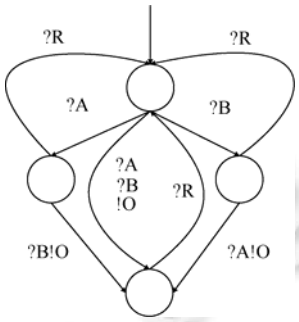

Fig. 3 State machine of ABRO

图 3 ABRO 的状态机表示

```
1:process ABRO=
2:(? boolean A, B, R;! boolean O;)
3:(|A^=B^=R
4:|A^=A_received
5:|A_received^=B_received^=after_R_until_O
6:|nR:=not R
7:|RT:=nR when R
8:|A_received:=RT default AR
9:|AT:=A when A
10:|AR:=AT default Adelay
11:|Adelay:=A_received $ init false
12:|BT:=B when B
13:|B_received:=RT default BR
14:|BR:=BT default Bdelay
15:|Bdelay:=B_received $ init false
16:|nO:=not O
17:|from_R_before_O:=nO default RR
18:|RR:=Re default after_R_until_O
19:|Re:=R when R
20:|after_R_until_O:=from_R_before_O $ init true
21:|O:=true when ABR
22:|ABR:=A_received when Arr
23:|Arr:=B_received when after_R_until_O|)
24:where boolean nR, nO, A_received, B_received,
from_R_before_O ,Adelay,Bdelay,AR,BR,RR,ABR,Arr,after_R_until_O,AT,BT,RT,Re; end;
```

Fig.4 SIGNAL program of ABRO

图 4 ABRO 的 SIGNAL 程序

3.2 时钟信息获取

通过对时钟方程进行解析后得到时钟等价类.图 5 展示了对时钟变量进行等价类划分的结果.等价类 *C*_2 为程序的根时钟.图 6 展示了对应的规约范式方程.可以看到,只有时钟 *C*_2 出现在方程的右侧,程序满足 endochrony 属性并且 *C*_2 为程序的根时钟.

C_2={^R,^A_received,^B_received,^after_R_until_O,^A,^B,^R,^nR,
^RR,^Adelay,^Bdelay,^from_R_before_O,^AR,^BR}
C_6={^RT,^R_true,^Re}
C_75={^A_received_default,^B_received_default,^RR_default}
C_13={^A_true,^AT}
C_26={^B_true,^BT}
C_77={^AR_default}
C_79={^BR_default}
C_84={^ABR,Arr_true}
C_92={^from_R_before_O_default}
C_82={^after_R_until_O_true,^Arr}
C_86={^ABR_true,^nO,^O}

Fig.5 Partition of clock equivalence classes for ABRO

图 5 对 ABRO 时钟等价类的划分

C_6=C_2∧R
C_75=C_2 \ C_6
C_13=C_2∧A
C_77=C_2 \ C_13
C_26=C_2∧B
C_79=C_2 \ C_26
C_82=C_2∧after_R_until_O
C_84=C_82∧Arr
C_86=C_84∧ABR
C_92=C_2 \ C_86

Fig.6 RNFS of ABRO

图 6 ABRO 的规约范式方程集合

3.3 EDG 的生成及并行任务划分

通过分析 ABRO 的数据流方程及数据依赖关系可以得到 EDG,如图 7 所示.需要说明的是,尽管图中 EDG 没有加入时钟信息,但在分析数据依赖关系时考虑了时钟关系,例如对于方程 ABR:=A_received when Arr,信号定义方程 ABR=A_received 同时依赖于 A_received 的值与 Arr 的值.此外,对于合并操作,这里将两个互斥的动作放在一个节点中,在一次迭代中,这两个动作将不会同时执行.得到的 EDG 将通过拓扑排序生成并行任务序列,并加入时钟信息,如图 8 所示.图中的实线箭头代表并行任务执行的线性关系,而同一个任务中的多个方程(实线方框中的椭圆方程)可以并行执行.

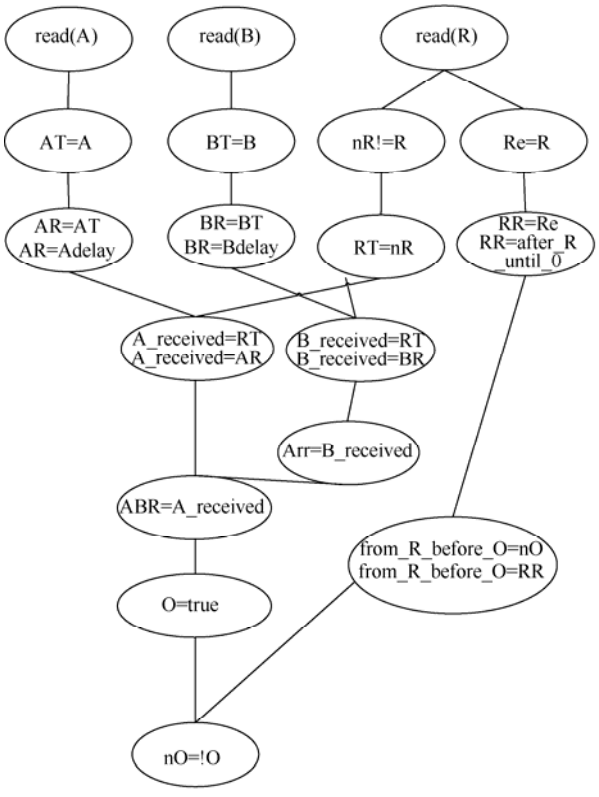

Fig.7 EDG of ABRO

图 7 ABRO 的 EDG 表示

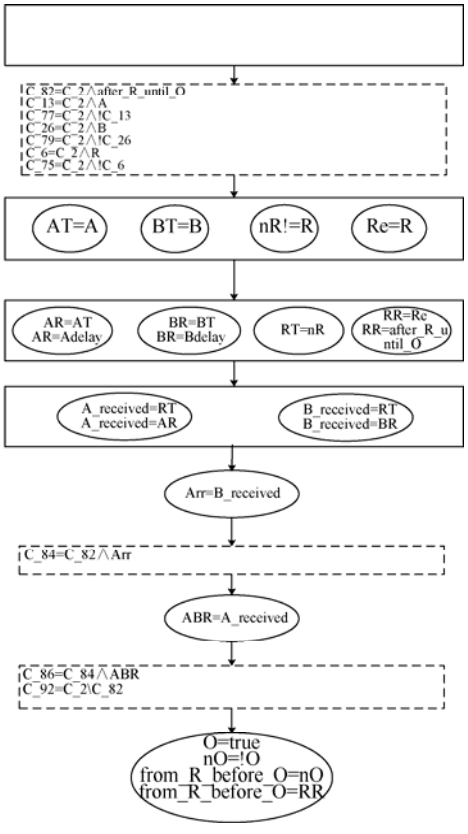

Fig.8 TaskList of ABRO

图 8 ABRO 的任务序列

3.4 OpenMP 并行仿真代码生成

图 9 给出了生成的 OpenMP 并行代码片段.可以看到,对 3 个输入信号 A、B、R 的读取可以同时进行,可生成的代码与其结构有一一对应关系.

```
1:int core(){
2:      int mark=0;
3:      #pragma omp parallel sections
4:      {
5:              #pragma omp section
6:              {
7:                      if(read_A()==EOF)
8:                              mark=1;
9:              }
10:             #pragma omp section
11:             {
12:                     if(read_B()==EOF)
13:                             mark=1;
14:             }
15:             #pragma omp section
16:             {
17:                     if(read_R()==EOF)
18:                             mark=1;
19:             }
20:     }
…
```

Fig.9 Fragment of OpenMP code

图 9 OpenMP 代码片段

3.5 代码的执行测试

我们基于 Eclipse 平台开发了 SIGNAL 代码生成工具.工具采用右键插件菜单的形式.通过在 Eclipse 下 Package Explorer 视图中在对应 SIGNAL 程序源文件位置点击右键选择进行代码生成,得到并行仿真代码.生成的仿真代码在 VS 2010 环境中进行测试,如图 10 所示.程序通过读取文本文件获得输入信号,并在输出信号得到值后向对应的文本文件进行输出.输入信号的文件中为一个序列,每一次迭代时,如果当前输入信号的时钟为真,则从序列中读取一个值.在 ABRO 程序示例中,输入信号有 A、B、R,输出信号为 O.对应的文本文件分别为 A.txt、B.txt、R.txt 以及 O.txt,3 个输入信号文件的信号值如图 11 所示.

表 6 给出了 ABRO 程序一次可能的执行结果.例如,对于逻辑瞬间 t_1,信号 A、B 同时为真而 R 为假,则输出信号 O 为真;而在逻辑瞬间 t_2,尽管 A、B 同时为真,但由于此时状态没有被重置,因此,此时 O 的值不存在;在逻辑瞬间 t_3,读入 R 的值为真,状态被重置(但此时,A、B 的值仍没有被读入,因此 O 的值不存在),因此在逻辑瞬间 t_4,A、B 同时读入后,O 取值为真.需要说明的是,由于程序仅在 O 取值为真时进行输出,因此不能输出 O 不存在时的值,即⊥.

Fig.10 Generated code in VS 2010 perspective

图 10 VS 2010 视图中的生成代码

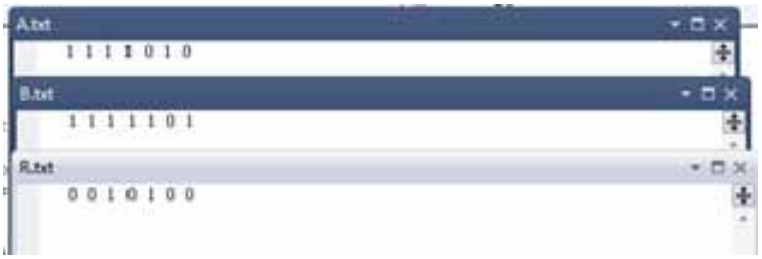

Fig.11 Example value of input signals

图 11 示例输入信号值

Table 6 A possible execution trace of ABRO

表 6 ABRO 的一次执行结果

信号	t_1	t_2	t_3	t_4	t_5	t_6	t_7
A	1	1	1	1	0	1	0
B	1	1	1	1	1	0	1
R	0	0	1	0	1	0	0
O	1	⊥	⊥	1	⊥	⊥	1

经过仿真得到 O 的输出如图 12 所示,其中的 3 个 1 与表 6 中 O 信号的输出相对应,可以看出生成的并行代码有效地对 ABRO 程序的功能行为进行了模拟执行.

Fig.12 Result of the execution

图 12 执行结果

4 相关工作介绍

针对 SIGNAL 的仿真代码生成,特别是多线程及分布式代码生成技术,有一些相关研究.但对跨平台执行的代码生产研究较少.

文献[26]介绍了面向 SIGNAL 的多线程代码生成技术,根据采取静态或动态调度方法可分为两种.静态调度方法中,编译器将根据调度图生成多个 cluster 代码块,并有一个主计算 cluster 用于进行迭代操作.每个 cluster 有自己的单根时钟树,读取输入后进入运行.在动态调度的分布式代码生成中,SIGNAL 程序中的每条可以并发执行的方程都将对应生成一个微线程(micro-thread).微线程通过 wait 等待输入信号,并在输出时发出 notify 信号.这样互不相关的过程可以并行执行.然而,生成的多线程仿真代码使用了特定的线程库,因此不能够跨平台执行.

在分布式或多核系统上,由于组件间的通信延迟等原因,导致组件间的信号不能满足同步关系,被称为全局异步局部同步(globally asynchronous locally synchronous,简称 GALS)系统.而此类系统不满足 endochrony 属性.为此,文献[27]提出了 weak endochrony 理论.就像名字所暗示的那样,weak endochrony 对于对象 SIGNAL 程序生成确定性代码所要求的属性更低:程序中可以存在多个根时钟,如果信号间满足 full-diamond 条件即可以生成确定性多线程代码.然而,检测程序是否满足 weak endochrony 属性是十分困难的.文献[28]提出了一种检测 weak endochrony 性质的方法,给出了从 SIGNAL 程序中找出可以构造出程序可能出现反应并且不会产生冲突的最小反应集合.如果此集合具有唯一性,则程序具有 weak endochrony 性质.但此种方法的缺点是需要首先将 SIGNAL 程序进行变换生成其无状态抽象,使得一些原本具有 weak endochrony 性质的程序丢失信息并导致其无状态抽象不满足该性质.文献[21]提出了基于有界模型检测以及基于 isochrony[29]属性的方法检查 weak

endochrony.然而,这两种方法也不能适用于所有 weak endochrony 程序.文献[30]提出了基于同步数据流依赖图(SFDG)的多线程代码生成方法,然而文献[30]中仍然采用文献[26]中使用的生成策略,并且没有给出 weak endochrony 属性的验证方法,也没有对方法的正确性进行验证.与前述的基于 weak endochrony 的多线程代码生成方法相比,尽管本文提出的方法目前主要支持 endochrony 属性的 SIGNAL 程序,但是通过 EDG 进行描述并将其进行并行任务划分也可以获得较好的并行度.

此外,文献[31]提出了一种从同步守护动作(synchronous guarded actions,一种同步语言中间表达形式)生成 OpenMP 并行代码的方法.同步守护动作首先被转换为依赖图(dependency graph),之后将从此依赖图中获取可同步执行的部分并最终生成 OpenMP 结构.然而,由于 SIGNAL 与同步守护动作在语法和语义,尤其是在时钟以及反应动作的语义方面差异很大,因此本文提出的方法与文献[31]有很大不同,尤其是在时钟分析方面,同步守护动作采用单时钟模型,不需要考虑时钟信息的获取,而本文则针对 SIGNAL 的多时钟模型,提出了获取时钟信息并将其加入到并行任务序列的方法.此外,在中间数据结构的设计上,与文献[31]提出的采用二分图的表示方法相比,本文采用的 EDG 可以更直观地描述程序全局的数据依赖关系.

与前述的研究相比,本文提出的方法主要有以下特点.

- 采用方程依赖图 EDG 可以更加全面地描述 SIGNAL 程序中方程间的数据依赖;基于拓扑排序方法,将可以并行执行的部分提取出来,仿真代码可以获得较好的并行度;
- 提出了一种面向 SIGNAL 同步规范的转换并行代码的方法,以 OpenMP 为对象,生成的并行仿真代码可以在多种系统平台上进行执行分析,具有更大的灵活性.

5 结论与未来工作

本文提出了一种针对 SIGNAL 同步规范的并行仿真代码生成方法.首先,基于布尔方程解析给出了提取 SIGNAL 程序时钟信息的时钟演算方法.之后,本文提出了方程依赖图 EDG 的概念,用于描述 SIGNAL 程序的全局数据依赖关系,并给出了基于拓扑排序方法对 EDG 进行并行任务划分的算法.最终并行任务序列被映射到 OpenMP 的 sections 结构.生成的 OpenMP 并行代码可以正确地仿真 SIGNAL 程序的功能行为,并且相对于针对特定平台线程库的仿真程序具有更大的灵活性.

未来工作将围绕以下 3 个方面进行:(1) 对并行任务划分方法进行优化,提高仿真代码并行度及执行效率.(2) 对代码生成过程作进一步的形式化,将流程进行模块分解,每一部分都将被形式化,并使用 Coq 或 Caml 等形式化语言对其正确性进行验证;在对每个模块进行验证之后,再对整体流程进行验证,以保证编译过程全程的正确性.(3) 对 weak endochrony 理论进行深入研究,研究验证 SIGNAL 程序是否满足 weak endochrony 的可行算法;在此基础上,研究从 weak endochrony 属性的 SIGNAL 程序到并行代码的生成方法.

致谢 来自法国 Toulouse Institute of Computer Science Research 的 Mamoun Filali 研究员与 Jean-Paul Bodeveix 教授在本文的写作过程中给予了很多重要的建议,在此对他们的帮助表示深深的感谢.

References:

[1] Knight JC. Safety critical systems: Challenges and directions. In: Proc. of the 24th Int'l Conf. on Software Engineering. Orlando, 2002. 547–550. [doi: 10.1145/581339.581406]

[2] Al DJ. MPI: A message-passing interface standard. Int'l Journal of Supercomputer Applications, 2009,20(2):179.

[3] The OpenMP API specification for parallel programming. http://www.openmp.org/specifications/

[4] Intel Thread Building Blocks. https://software.intel.com/en-us/forums/intel-threading-building-blocks/

[5] Benveniste A, Gérard B. The synchronous approach to reactive and real-time systems. Proc. of the IEEE, 1991,79(9):1270–1282. [doi: 10.1109/5.97297]

[6] Benveniste A, Caspi P, Edwards SA, Halbwachs N, Le Guernic P, De Simone R. The synchronous languages 12 years later. Proc. of the IEEE, 2003,91(1):64–83. [doi: 10.1109/JPROC.2002.805826]

[7] Berry G, Gonthier G. The ESTEREL synchronous programming language: Design, semantics, implementation. Science of Computer Programming, 1992,19(2):87−152. [doi: 10.1016/0167-6423(92)90005-V]

[8] Halbwachs N, Caspi P, Raymond P, Pilaud D. The synchronous data flow programming language LUSTRE. Proc. of the IEEE, 1991,79(9):1305−1320. [doi: 10.1109/5.97300]

[9] Le Guernic P, Gautier T, Le Borgne M, Le Maire C. Programming real-time applications with SIGNAL. Proc. of the IEEE, 1991, 79(9):1321−1336. [doi: 10.1109/5.97301]

[10] Schneider K. The synchronous programming language QUARTZ. Technical Report, Internal Report 375, Kaiserslautern: Department of Computer Science, University of Kaiserslautern, 2009.

[11] Yu HF, Ma Y, Thierry G, Loïc B, Jean-Pierre T, Le Guernic P, Yves S. Exploring system architectures in AADL via Polychrony and SynDEx. Frontiers of Computer Science, 2013,7(5):627−649. [doi: 10.1007/s11704-013-2307-z]

[12] Polychrony. http://www.irisa.fr/espresso/Polychrony/

[13] Kirsch CM. Principles of real-time programming. In: Proc. of the 2002 Conf. on Embedded Software. Berlin, Heidelberg: Springer-Verlag, 2002. 61−75. [doi: 10.1007/3-540-45828-X_6]

[14] Jantsch A, Sander I. Models of computation and languages for embedded system design. In: Proc. of the IEEE of Computers and Digital Techniques—IET. 2005. 114−129. [doi: 10.1049/ip-cdt:20045098]

[15] Potop-Butucaru D, de Simone R, Jean-Pierre T. The synchronous hypothesis and synchronous languages. 2015. http://pdf.aminer.org/000/213/379/modeling_distributed_embedded_systems_in_multiclock_esterel.pdf

[16] Gamatie A. Designing Embedded Systems with the Signal Programming Language: Synchronous, Reactive Specification. Springer Publishing Company, Incorporated, 2010. [doi: 10.1007/978-1-4419-0941-1]

[17] Le Guernic P, Jean-Pierre T, Le Lann JC. Polychrony for system design. Journal for Circuits, Systems and Computers, 2003,12(3): 261−303. [doi: 10.1142/S0218126603000763]

[18] Besnard L, Gautier T, Le Guernic P. Signal v4-Inria version: Reference Manua. 2004. http://www.irisa.fr/espresso/Polychrony/document/V4_def.pdf

[19] Yang Z, Bodeveix JP, Filali M. A comparative study of two formal semantics of the SIGNAL language. Frontiers of Computer Science, 2013,7(5):673−693. [doi: 10.1007/s11704-013-3908-2]

[20] Hu K, Zhang T, Yang Z. Multi-Threaded code generation from signal to OpenMP. Frontiers of Computer Science, 2013, 7(5):617−626. [doi: 10.1007/s11704-013-3906-4]

[21] Jean-Pierre T, Ouy J, Gautier T, Besnard L, Guernic PL. Compositional design of isochronous systems. Science of Computer Programming, 2012,77(2):113−128. [doi: 10.1016/j.scico.2010.06.006]

[22] Benveniste A, Caillaud B, Le Guernic P. From synchrony to asynchrony. In: Proc. of the CONCUR. 1999. 162−177. [doi: 10.1007/3-540-48320-9_13]

[23] Hu K, Zhang T, Yang Z, Tsai W. Simulation of real-time systems with clock calculus. Simulation Modelling Practice and Theory, 2015,51:69−86. [doi: 10.1016/j.simpat.2014.10.010]

[24] Maffeïs O, Le Guernic P. Combining dependability with architectural adaptability by means of the SIGNAL language. In: Static Analysis. Berlin, Heidelberg: Springer-Verlag, 1993. 99−110. [doi: 10.1007/3-540-57264-3_32]

[25] Plotkin G, Stirling CP, Tofte M. The foundations of Esterel. In: Proof, Language, and Interaction: Essays in Honour of Robin Milner, Vol.1. MIT Press, 2000. 425−454.

[26] Besnard L, Gautier T, Jean-Pierre T. Code generation strategies in the Polychrony environmen. 2009. http://hal.inria.fr/docs/00/37/24/12/PDF/RR-6894.pdf

[27] Potop-Butucaru D, Caillaud B, Benveniste A. Concurrency in synchronous systems. Formal Methods in System Design, 2006, 28(2):111−130. [doi: 10.1007/s10703-006-7844-8]

[28] Potop-Butucaru D, Simone RD, Sorel Y, Talpin JP. From concurrent multi-clock programs to deterministic asynchronous implementations. Fundamenta Informaticae, 2011,108(1):91−118.

[29] Benveniste A, Caillaud B, Le Guernic P. Compositionality in dataflow synchronous languages: Specification and distributed codegeneration. Information and Computation, 2000,163(1):125−171. [doi: 10.1006/inco.2000.9999]

[30] Jose BA, Shukla SK, Patel HD, Talpin JP. On the deterministic multi-threaded software synthesis from polychronous specifications. In: Proc. of the 6th ACM & IEEE Int'l Conf. on Formal Methods and Models for Co-Design (MEMOCODE 2008). 2008. 129–138. [doi: 10.1109/MEMCOD.2008.4547700]

[31] Baudisch D, Brandt J, Schneider K. Multithreaded code from synchronous programs: Extracting independent threads for OpenMP. In: Proc. of the Conf. on Design, Automation and Test in Europe. European Design and Automation Association, 2010. 949–952.

胡凯(1963－),男,湖南长沙人,博士,教授,主要研究领域为分布式系统,嵌入式实时系统.

张腾(1989－),男,硕士,主要研究领域为形式化方法,安全关键实时系统设计.

尚利宏(1971－),男,博士,副教授,CCF 专业会员,主要研究领域为嵌入式系统.

杨志斌(1982－),男,博士,CCF 专业会员,主要研究领域为形式化方法,安全关键实时系统.

Jean-Pierre TALPIN(1964－),男,博士,教授,博士生导师,主要研究领域为形式化方法,嵌入式系统.